Long live the HP-11C !

Valentin Albillo (Ex-PPC member #4747)

It's been 20 years since I was presented with my HP-11C, and it seems as if no time has passed by at all. Well, at least not for my trusty 11C, which still is as mint as ever, despite being frequently used and abused. As for myself, I can't say the same, those 20 years have exacted their toll on my ever youthful self indeed, but then I'm not one of its slim kind.

Now, I've always thought that the 11C, like all machines in the legendary Voyager series, were well ahead of their time, almost timeless. There's nothing right now which compares. Talk about their ultra-elegant style, those slim bodies which can fit comfortably in most every pocket. Talk about their ruggedness, they felt as solid as a brick. Talk about the keyboard, with those real, positively-clicking keys, with molded, unerasable helvetica lettering on them. Talk about their nearly unbelievably low power consumption, which made batteries last for years and years and years, a real "Duracell bunny" of sorts. And of course, talk about their functionality. The best Voyager models have an incredible amount of computing power in those small bodies, which has taken a lot of years to surpass, and never again in such marvelous hardware.

To illustrate this, and to commemorate the 11C as it deserves, here is a couple of programs I wrote 20 years ago, which suitably show its capabilities to the max. The first one, Exhibit A, is a short program which performs an unusual mathematical task, and you'll find it included right here, in this article. The other one, Exhibit B, allows your 11C to play a challenging, non-trivial board game against you, and furthermore, it can play on boards of *any* size NxN ! But you'll have to wait till the next issue to test your brains against it, and you won't find it easy to beat at all, I promise ...

Exhibit A: Summation of infinite, alternating series

Given an arbitrary, user-defined infinite series whose terms are alternately positive and negative, this program will compute accurately its infinite sum very quickly, even if the series convergence is dismayingly slow. More so, it will even succeed for *divergent* series !

For example, consider the infinite series S = 1 - 1/2 + 1/3 - 1/4 + 1/5 - ...
= Ln(2) = 0.693147181+

Trying to compute its infinite sum using a loop to add a suitable number of terms would require *incredible* amounts of time, even on the fastest computer !. For

instance, you would need to add more than 2000 terms just to get a mere 3 correct decimals, and many millions of terms to achieve 7 or more digits correct. The rounding errors after adding so many terms would also badly affect the maximum accuracy obtainable.

In contrast, this program allows your HP-11C to compute that sum or any other you care to specify accurate to 9 or 10 digits, in one minute or less. See examples:

Program listing:

```
001  LBL A     026    /      051  STO I     076  ISG
002  CF 0      027  FRAC     052  STO .2    077  FIX 4
003  STO .1    028  X#0      053  LBL 0     078  RCL .1
004  X<>Y      029  SF 0      054  RCL(i)    079  RCL I
005  STO .0    030  CLX      055  ISG       080  X<=Y
006    1       031  STO I     056  FIX 4     081  GTO 4
007  STO 8     032  LBL 2     057  STO-(i)   082  RCL 9
008    0       033  RCL 8     058  RCL .1    083  RTN
009  STO 9     034  GSB B     059  RCL I     084  LBL B
010  STO I     035  STO(i)    060  X#Y
011  LBL 1     036  ISG       061  GTO 0     84 steps
012  GSB B     037  FIX 4     062  RCL .2
013  RCL 8     038    1       063  STO I     Labels:
014  STO-8     039  STO+8     064  X=0       A,B,0,1,2,4,6
015  STO-8     040  RCL .1    065  GTO 4
016    *       041  RCL I     066    1       Registers:
017  STO+9     042  X<=Y      067  GTO 6
018  ISG       043  GTO 2     068  LBL 4     0 thru .2, I
019  FIX 4     044    2       069  RCL(i)
020  RCL .0    045  F? 0      070  RCL 8     Flags:
021  RCL I     046  CHS       071    /
022  X<=Y      047  STO 8     072  STO+9     0
023  GTO 1     048  ABS       073    2
024  STO 8     049  LBL 6     074  CHS       Ang.Mode: Any
025    2       050   -        075  STO*8
```

Usage instructions:

1) Define the i-th term of the series under label B: press

GTO B, switch to PRGM mode, key in the keystroke sequence that computes the i-th term as a function of i, and end it with RTN. Then switch back to RUN mode.

For definition purposes, the index "i" is on the X register when B is called. The index “i” assumes the values 0,1,2,3,…, so the very first term of your series is the one corresponding to i=0. You can use available registers .3 (watch out, it's .3, not 3) onwards in your definition. Also, do not define a sign for your i-th term, it's automatically assumed that its sign alternates between positive and negative.

2) Enter into the stack the number of terms to sum initially (PSum) and the number of differences to compute (NDif), and, in USR mode, press A (or out of USR mode, press GSB A or shift-A):

PSum, ENTER, NDif, A (or GSB A if not in USR mode)

The program runs and after a short while, it will stop with the computed infinite sum of the series on the display.

3) To try another values of PSum or NDif go to step (2) above. To sum a different series, go to step (1) above [don't forget to delete the previous definition of the i-th term from program memory, except LBL B itself, before keying in the new one]

Notes:

- Using values of PSum = 7 and NDif = 7 is recommended for good accuracy/speed. You may want to try other combinations, but these are good default values.

- PSum must be an integer >= 0. It may be larger than 7, say 10 or more, if desired

- NDif must be an integer between 1 and 7, both included. It can't be > 7 nor < 1

Now some examples, we'll assume USR mode is active, and FIX 9:

Example 1:

Find the sum of S = 1 -1/2 + 1/3 - 1/4 + 1/5 - ...

- we define the i-th term = 1/(1+i):

GTO B, P/R, 1, +, 1/x, RTN, P/R

- we'll use PSum = 10, NDif = 7 for maximum accuracy:

10, ENTER, 7, A

- after just one minute, we get the sum = 0.693147182 in the display. The theoretically correct value is Ln(2) = 0.693147181, so we've got 9 decimals accuracy.

Example 2:

Find the sum of S = 1/0.23 - 1/0.24 + 1/0.25 - 1/0.26 + ...

- we define the i-th term = 1/(0.23+0.01*i) : [previous example definition assumed deleted from program memory]. For extra speed, let's use previously stored constants in R.3 and R.4 (notice they are R.3 and R.4, not R3 and R4):

GTO B, P/R, RCL .3, *, RCL .4, +, 1/x, RTN, P/R

- store the constants: 0.01, STO .3, 0.23, STO .4

- let's use PSum = 6, NDif = 6:

6, ENTER, A

- after only 50 seconds, we get the sum = 2.221127525 in the display. The theoretically correct value is 100*INTEGRAL(0,1,x^22/(1+x).dx) = 2.221127525, so we've got maximum possible accuracy, 10 exact digits.

Example 3:

Find the "sum" (!) of the *divergent* series S = 1 -2 +3 -4 +5 -6 + ...

- we define the i-th term = 1+i [previous example definition assumed deleted from program memory]:

GTO B, P/R, 1, +, RTN, P/R

- we'll use PSum = 0, NDif =1:

0, ENTER, 1, A

- after less than 10 seconds, your 11C achieves the impossible, and sums a divergent series ! Actually, we get the "sum" = 0.250000000 in the display. Of course, this series is divergent so it has no sum in the normal sense. What the program computes

in this case is called the "Eulerian Sum" of the function. Why 0.25 ? Consider this Taylor Series Expansion:

$$1/(1+x)^2 = 1 -2x + 3x^2 -4x^3 +5x^4 - ...$$

which is only valid for ABS(X)<1. However, if we make x =1 in both sides of the above identity, we get:

$$1/(1+1)^2 = 1/4 = 0.25 = 1 -2 +3 -4 +5 -6 + ...$$ bizarre, isn't it ?

That's all. You may want to try the program in some very slow converging examples of your own, for instance, try 1-1/4+1/9-1/16+1/25- ..., and compare with the exact value. Or if you're feeling adventurous, try 1-1/2+1/3-1/5+1/7-1/11+1/13- ..., where the denominators are the i-th prime, you just need to insert an i-th prime function under LBL B !

As a final, sad remark, it's a real pity that we won't ever be able to see such wonderful machines being marketed again, except for the most successful Voyager calculator ever, the HP-12C. Then again, our existing Voyagers are sure to serve us for long decades to come. Long live the 11C !